AI 응용시스템의 이해와 구축

7강.

출석.

Final Project Discussion

잘 준비 되어 가시나요?

- 첫 draft 제출: 4/23 (중간고사 일정)

중간고사

중간고사 일정: 4/23

별다른 announcement 없으면 비대면.

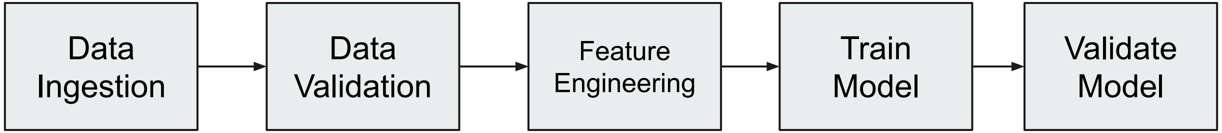
- 범위: 오늘 (7강) 수업내용까지
- Factual components
- Case-study
    - Final project와 유사.
    - 주어진 비즈니스 문제를 모델링 문제로 디자인, 모델링 파이프라인 디자인 등.

AutoML

AutoML

Automated Machine Learning (AutoML)

- ML 경험이 풍부하지 않은 개발자도 최적화된 모델을 찾도록 도와주는 테크닉



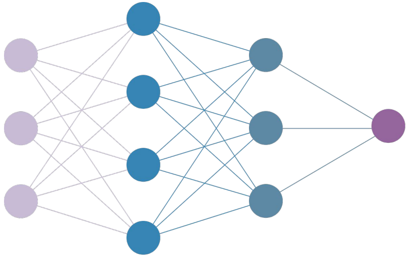
End 2 End ML 파이프라인 전체를 자동화하는 테크놀로지.

Neural Architecture Search (NAS)

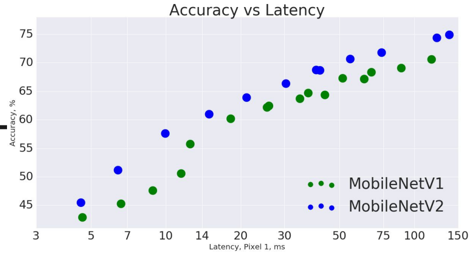
Search strategy를 사용해 Search space에서 캔디데잇을 선택

Architecture Search Space

Search Strategy



Candidate Model



Performance Estimation Strategy

AutoML의 한 종류: 모델 아키텍쳐 선택의 자동화
[Neural Architecture Search: A Survey, Elsken et al. 2019]

Chain-structured Search Space

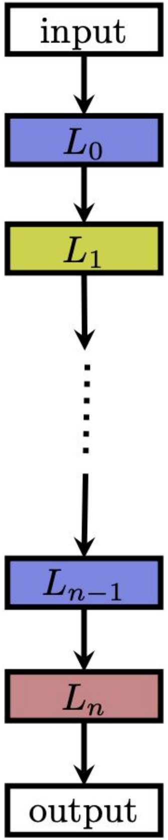
각 L0, L1, 등은 모델의 layer를 뜻함.

각기 다른 색상은 해당 layer의 op type 을 의미함.

Chain-structured Networks:

- # of layers
- Type of operation per layer: pooling, conv, ..
  - Hyperparams: # of filters, conv size, # of units

Macro-search



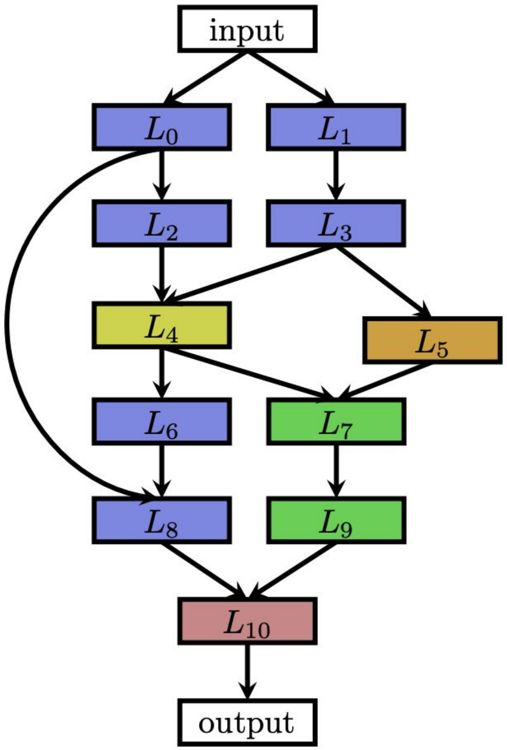
Macro-search

Complex-structured Search Space

Custom component를 사용해 더 복잡도 있는 네트워크도 형성 가능

- Layer: defined as a function
- Multi-branch network 형성할 수 있다.
  - Branch op
  - Skip connection

각 custom component는 hand-crafted로 형성해서 search에 이용.



Micro-search

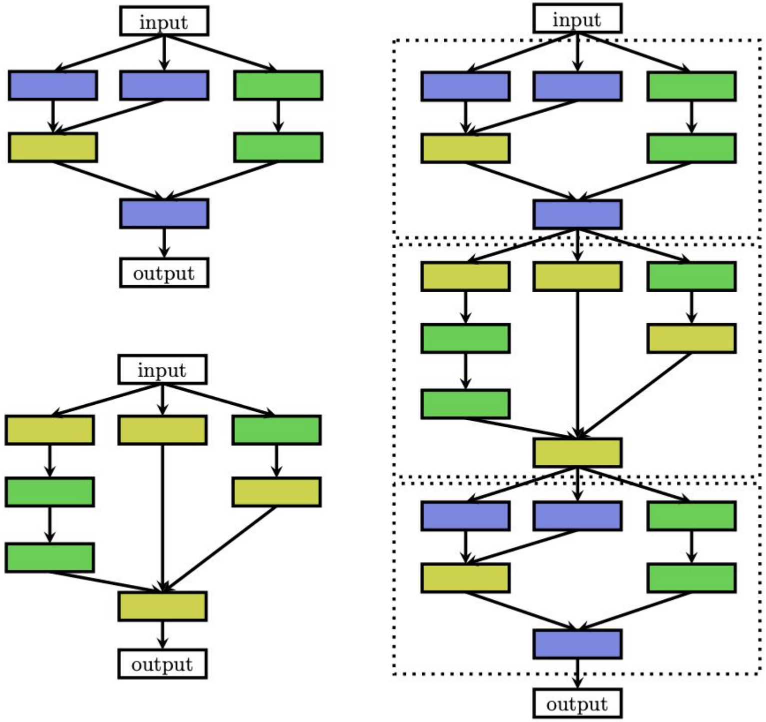
Transferrable Architecture Search

Network Motif: 상기 메소드중 custom layer들을 hand-crafted 형식으로 생성하기보다, 해당 Cell을 탐색해서 발견.

- **Normal Cell:** 이미지의 dimension을 줄이지 않는 형식
- **Reduction Cell:** feature map을 통해 dimension을 줄이는 형식

Cell들이 탐색되면 그들을 조합하여 Search Space를 표현함. (오른쪽)

Zoph et al, "Learning transferable architectures for scalable image recognition", CVPR 2018

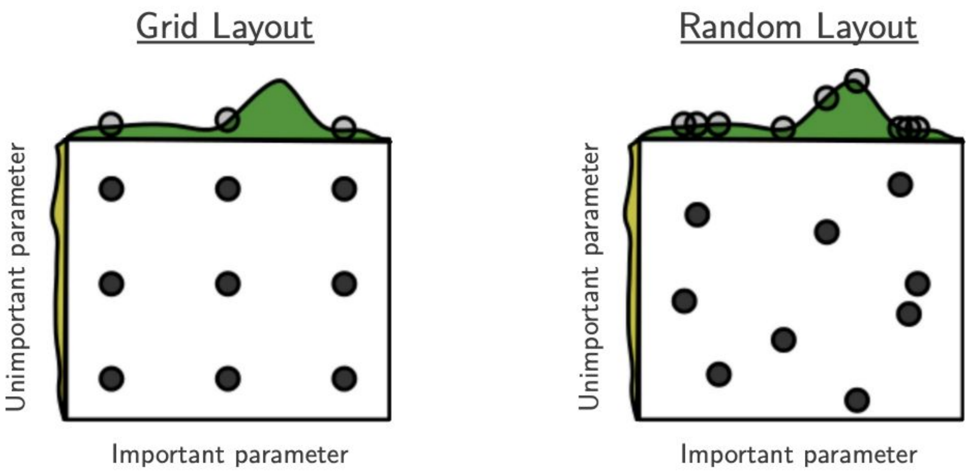


Search Strategy

네트워크 검색 알고리즘은 크게 몇가지가 있다.

1. Grid Search
2. Random Search
3. Bayesian Optimization
4. Evolutionary Algorithm (Genetic Algorithm)
5. Reinforcement Learning

Simple Search



Random search가 grid보다는 효과적.

[Bergstra, Bengio, “Random search for hyper-parameter optimization”, JMLR (2012)]

- Grid Search
  - Search space안의 모든 network configuration을 하나씩 exhaustive search 하는 방법.
- Random Search
  - Search space 안의 network configuration을 랜덤하게 추출하여 검색하는 방법.

Search space자체가 작을 경우 둘 다 효과적임.

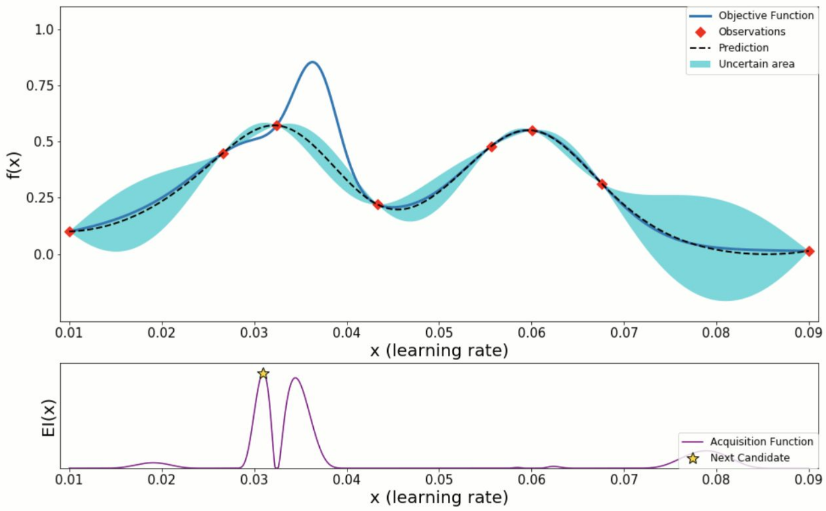
Search space가 클 경우?

- 당연히 많은 시간이 걸림.

Bayesian optimization network search

인풋 x가 주어졌을때, 목적함수 f 에 대해 함수값 f(x)를 최대로 만드는 해를 찾는것이 목적.

- f(x)와 hyperparameter의 pair로 Surrogate Model (대체 모델)을 만들고
- 순차적으로 hyperparameter를 업데이트해가며 최적의 hyperparam을 찾는다.



빨간점: 현재까지 관측한 값
점선: 빨간점들을 대상으로 예측한 estimation
파란선: groundtruth f(x)
녹색영역: f(x)가 있을만한 confidence bound.

Acquisition function: 다음 hyperparam 입력값을 추천할 때 사용

Evolutionary Algorithms

Evolutionary algorithm은 다음과 같은 스텝들로 구성된다.

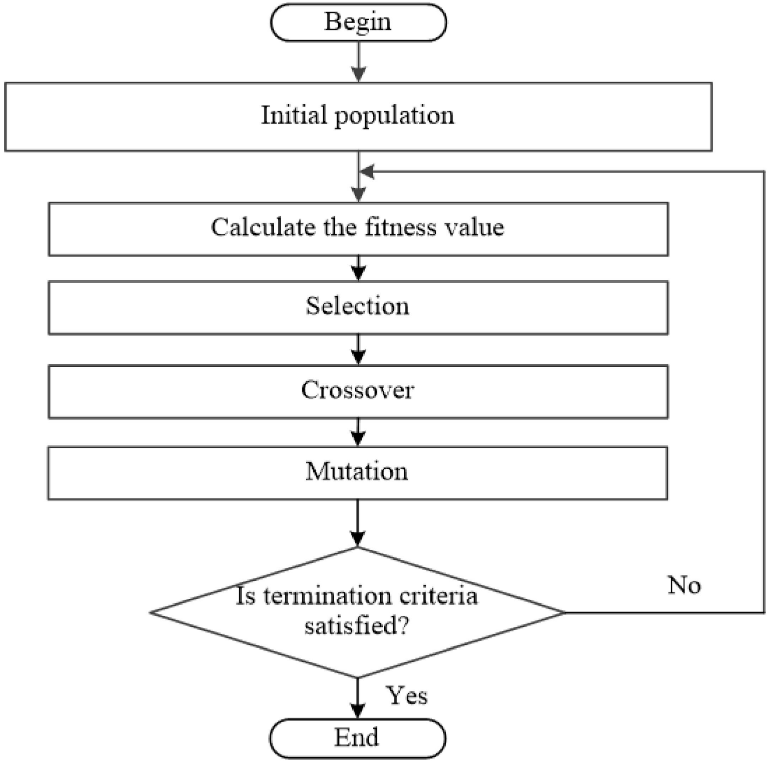
Initialization:
처음 시작할 network candidate들을 생성

Random selection

Selection:
Candidate pool에서 뛰어난 네트워크들을 선택

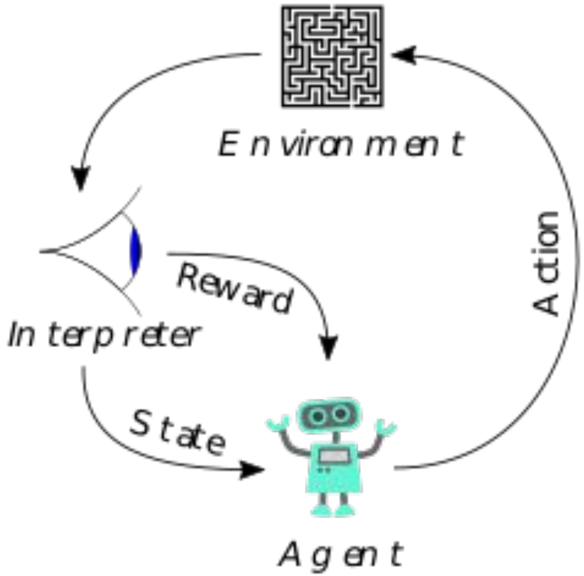
Cross over:
선택된 candidate중에서 hybrid (offspring) candidate을 생성

Mutation:
랜덤한 layer를 addition / removal

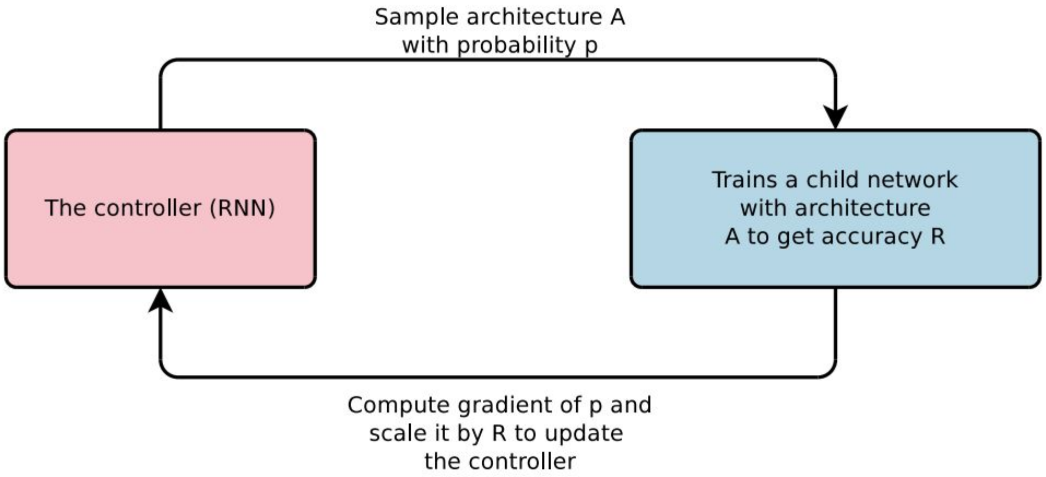


Reinforcement Learning (강화학습)

- RL: Agent의 골은 reward 를 최대화 하는것
- Search space에서 가능한 option candidate 을 선택
- Performance estimation strategy가 reward를 정의함.



RL for NAS



RNN을 string으로 인코딩:

- character당 network layer 정의

CIFAR-10 네트워크에서 해당방법으로SOTA 도달.

Zoph, Le, "Neural architecture search with reinforcement learning", ICLR 2017

Performance Estimation Strategy

탐색된 candidate이 있을때 얼마나 performant한지 측정하는 방법.

Validation accuracy를 사용한다면?

- Computation heavy
- Time consuming

Performance Estimation Strategy

탐색된 candidate이 있을때 얼마나 performant한지 측정하는 방법.

- Lower fidelity
- Learning Curve Extrapoloation
- Weight Inheritance / Network Morphism

Lower Fidelity Estimates

Goal: 학습에 걸리는 시간을 줄임

- Train data의 일부분만 사용해서 학습
- Vision모델의 경우: Lower resolution image 사용
- Filter / cell 갯수 줄이기

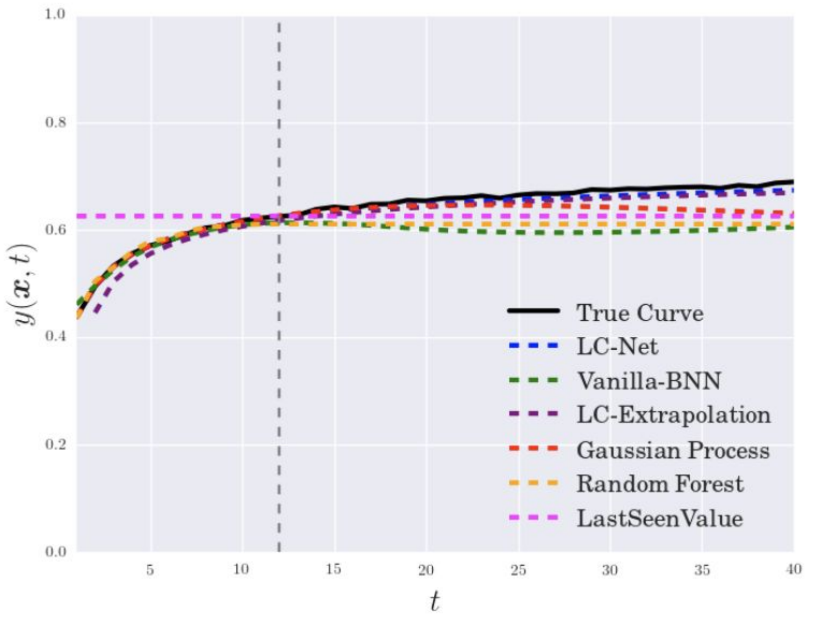
cost는 줄지만 해당 모델의 퍼포먼스 감소.

Candidate간에 상대적 랭킹이 lower fidelity를 적용하여도 유지된다면 사용가능.

그러나 많은 경우 relative ranking이 유지되지 않음.

Learning Curve Extrapolation

- Learning curve (Accuracy 커브)를 예측하여 사용.
- 초기의 learning curve를 사용하여 extrapolation을 진행
- Bayesian Optimization: 때로는 surrogate model을 사용하기도 함.



Klein et al., "Learning curve prediction with bayesian neural networks", ICLR 2017

Weight Inheritance / Network Morphism

- 새로운 candidate model 의 weight을 미리 학습된 다른 모델들의 weight로 시작
    - “Transfer learning”과 흡사
- Network morphism을 사용하여 parent model에서 모델을 변경 후 해당 parent model들의 historical metric에 따라 bayesian으로 추정한다.
    - Evolutionary algorithm의 mutation step과 흡사

- Network Morphism Tuner

Model Optimization (모델 최적화)

Model Optimization

- 무거운 모델을 deploy 할 때 고려해야 하는 점:
  - Inference latency: 추론에 얼마나 시간이 걸리나?
  - Model size: 메모리 사이즈
  - Energy consumption: 추론을 위한 에너지(배터리) 사용도

- 모델 최적화를 통해 위 이슈들을 발전시킴에따라 model quality (정확도)와 trade-off

Break

Optimize your TensorFlow Lite Models

ML / AI

Optimize your TensorFlow Lite Models

AI/ML

TensorFlow Lite is Google's production ready framework for deploying ML models on edge devices and embedded systems

Play Store

GMail

Search

Google Play

ML Kit

Photos

Youtube

Assistant

Firebase

Android Studio

4B+

Devices

What we enable

Text

Classification
Prediction
Q&A

Speech

Recognition
Text to Speech
Speech to Text
Hotword detection

Image

Object classification
Object detection
Gesture recognition
Facial modelling
Segmentation
Clustering
Compression
Super resolution
OCR
Style transfer

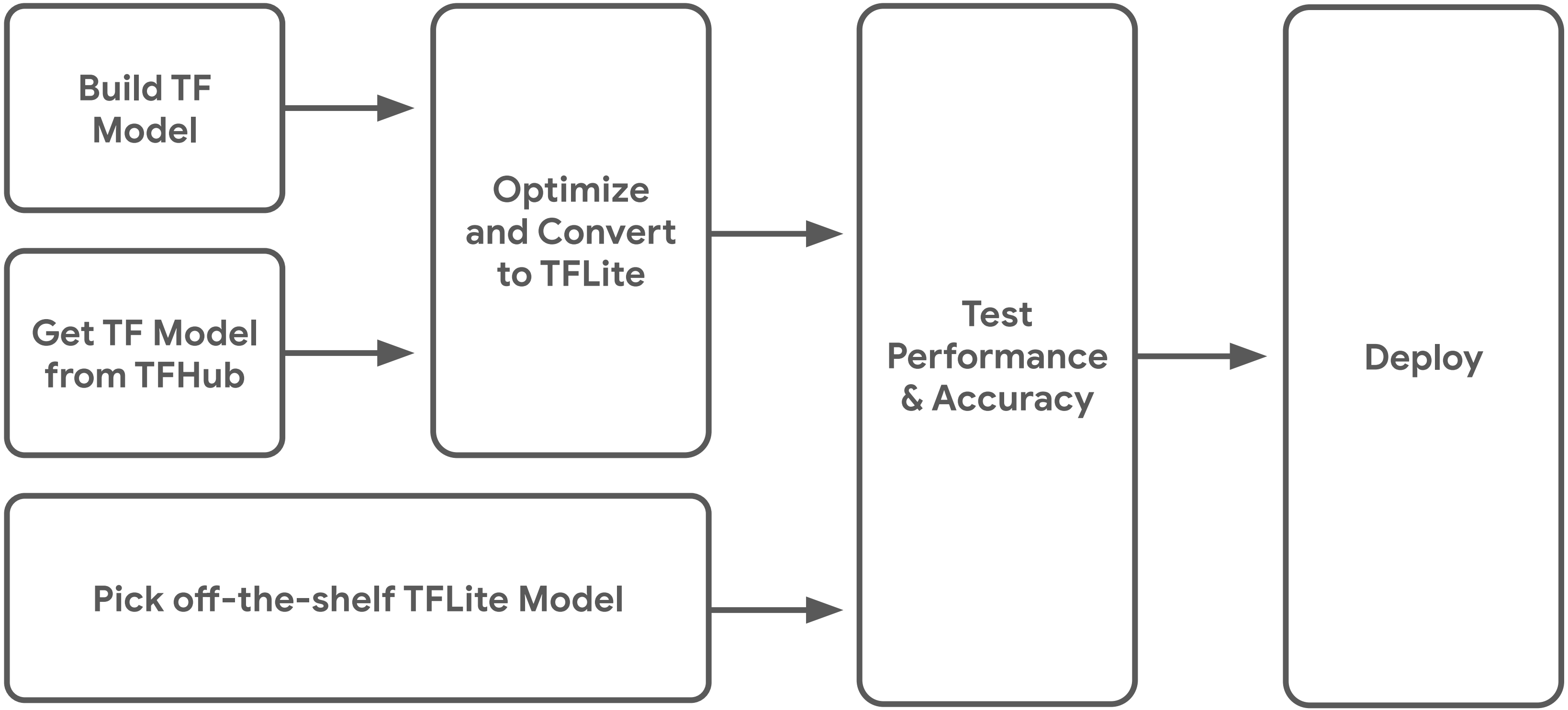
Audio

Translation
Voice synthesis
Music detection
Noise detection

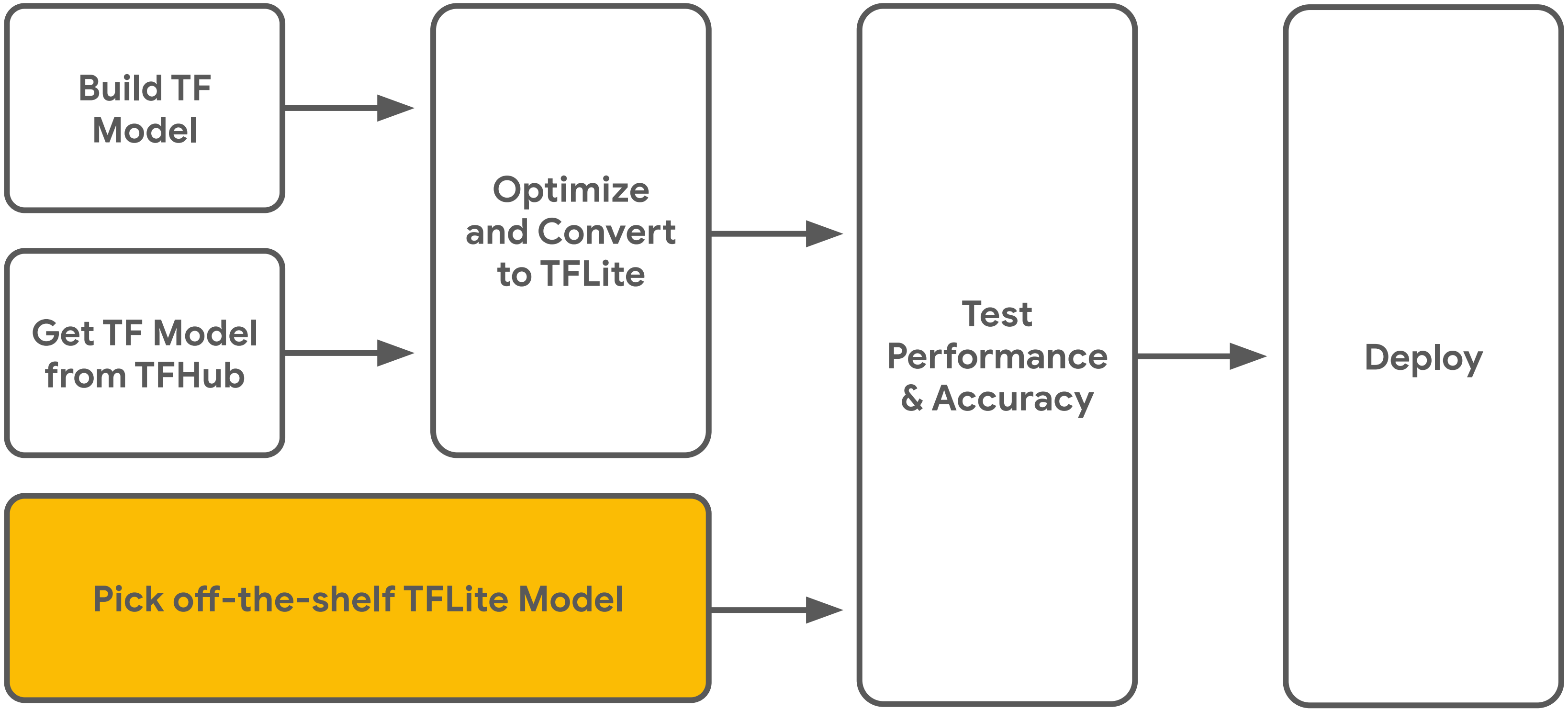
Content

Video generation
Text generation
Audio generation
Recommendation

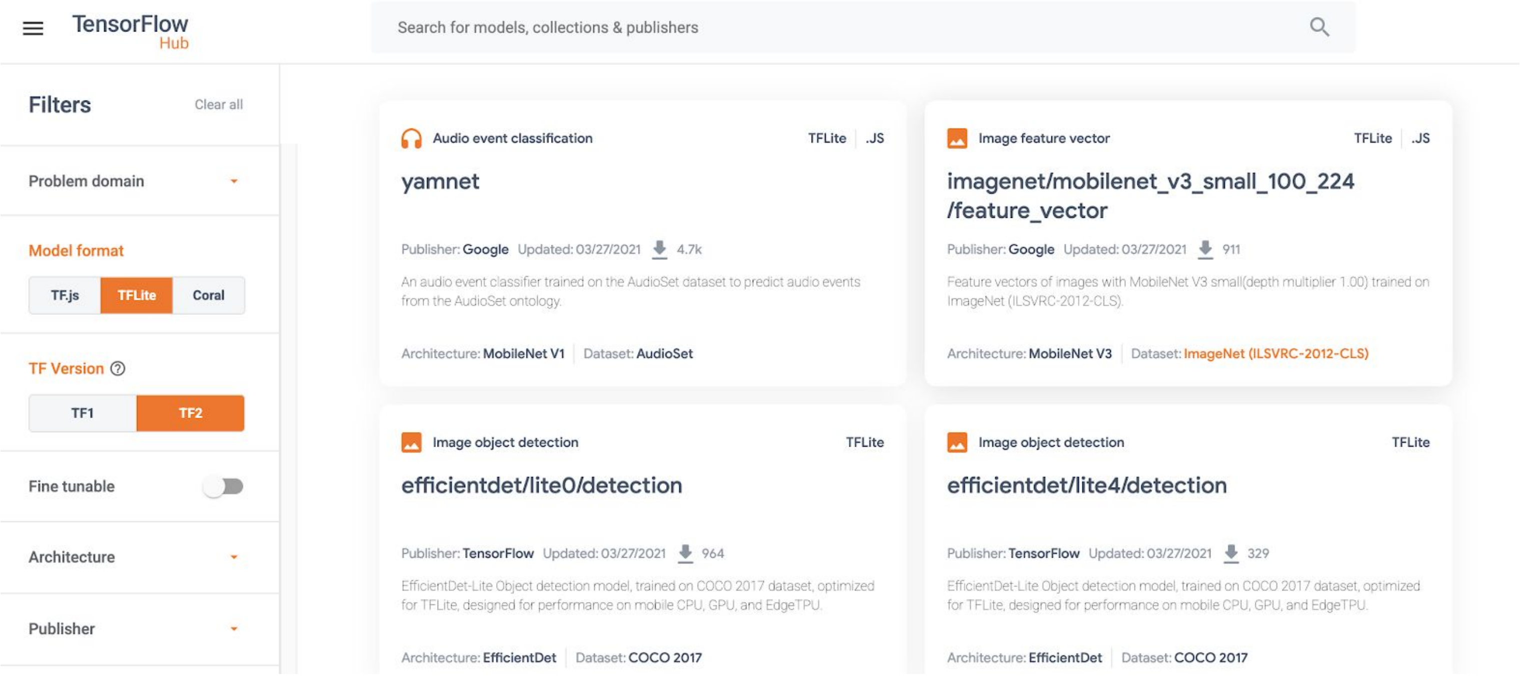
Workflow at a glance



Workflow at a glance

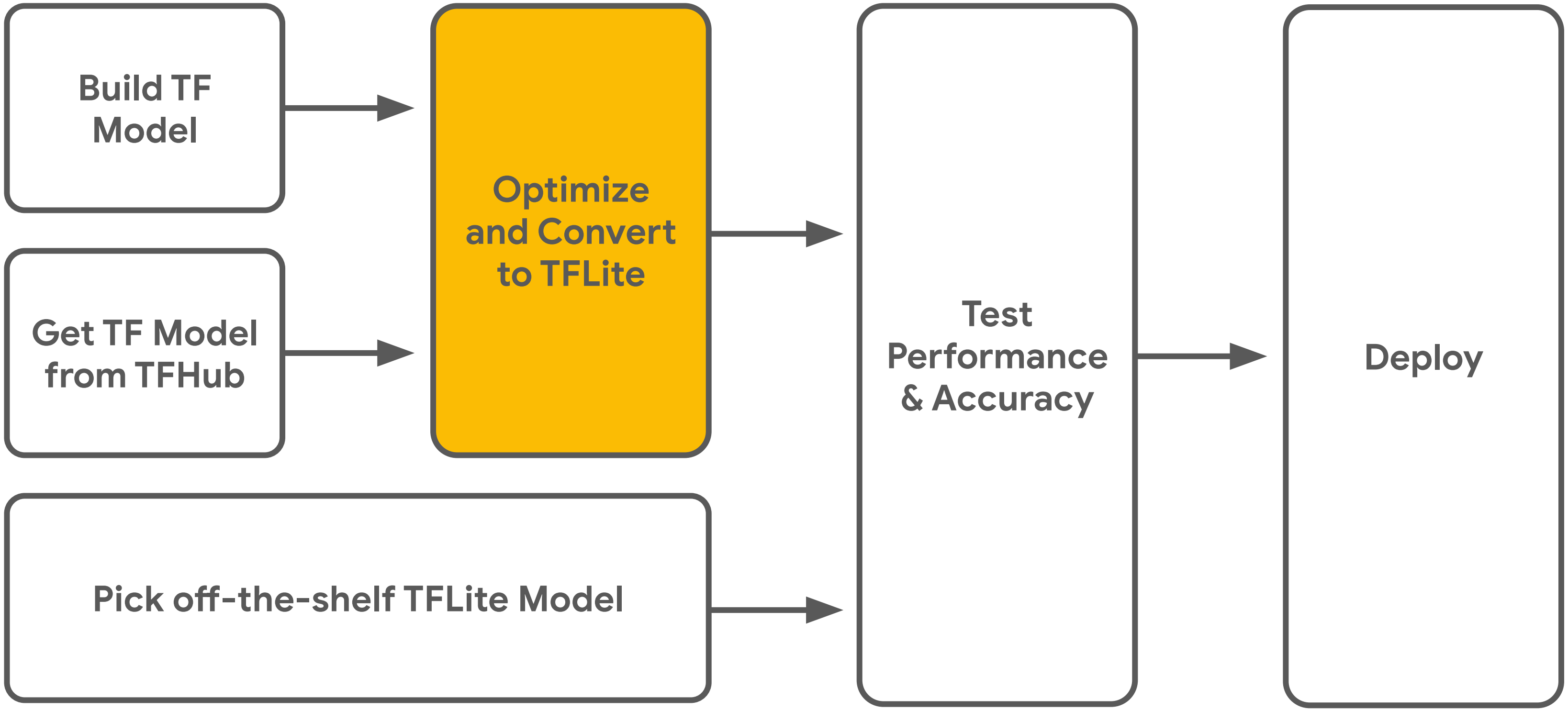


Model discovery - TensorFlow Hub



tensorflow.org/hub

Workflow at a glance



Model Optimization Toolkit is a suite of tools and techniques for **optimizing** machine learning models to run **smaller** and **faster.**

tensorflow.org/model_optimization

Quantization

Quantization transforms an ML program into an **approximated** representation with lower precision operations.

- Reduce precisions for **static** values (weights)
- Reduce precisions for **dynamic** values (activations)
- Modify / add / remove operations

We offer **Post-training** API and **During-training quantization*** APIs.

*During-training quantization: also known as Quantization-Aware Training

Quantization

Quantization (양자화):

- 뉴럴넷에서 계산 (computation)과 텐서(storage) 를 floating point보다 낮은 bitwidth로 진행하는 테크닉.
- 이 테크닉으로 float32가 아닌 integer로 계산 가능

- 따라서 model representation을 float model보다 작게 가능
  - Smaller model size

- 많은 하드웨어 플랫폼이 integer vector computation을 지원
  - Faster execution (inference) time!

Quantization

Quantization (양자화):

- Inference Only: Quantization은 inference 에만 영향이 가며, 학습 속도와는 관련이 없습니다.
- 모든 layer가 quantize되지는 않음:
    - 각 layer op마다 지원이 안되는 경우도 존재.
- 따라서 모델마다 quantization에 따른 loss가 다름.

Formula:

- Quant: float32 를 int8등으로 변경할 때 사용하는 operation
- Dequant: int8을 다시 float32로 변경해야 할때 사용.

Quantization Process

α와 β를 사용해서 원하는
레인지로 quantization 실행

- Quantization 은 floating point 에서 b-bit int로 맵핑

$$x \in [\alpha, \beta] \Longrightarrow x_q \in [\alpha_q, \beta_q]$$

- 따라서 다음과 같은 프로세스가 성립된다 (**De-quant**):

$$x = s(x_q - z)$$

z = integer (**zero-point**)
s = float (**scale**)

- 이 프로세스를 거꾸로 역산하면 (**Quant**):

$$x_q = \text{round}(\frac{1}{s}x) + z$$

따라서 스케일(s)과 제로포인트
(z)를 구하면 quant/dequant를
실행할 수 있다.

Quantization Process

- 필요한 range (alpha~beta사이) 를 얻기 위해 c와 d를 계산해보면:

$$\alpha = s(\alpha_q - z) \qquad \beta = s(\beta_q - z)$$

- Linear system을 풀어보면:

$$s = \frac{\beta - \alpha}{\beta_q - \alpha_q} \qquad z = \mathrm{round}\left(\frac{\beta\alpha_q - \alpha\beta_q}{\beta - \alpha}\right)$$

Quantization Process

- 마지막으로, 실제 모델에서는 alpha와 beta range를 명확히 guarantee하기 어려울 수 있다.
    - 예를들어, inference data에는 존재하나 train set에는 존재하지 않는 데이터가 나올수 있기 때문.
- 따라서, 실제 quantization process는 “clipping” 계산이 들어갈 수 있다.

$$x_q = \mathrm{clip}(\mathrm{round}(\frac{1}{s}x) + z, \alpha_q, \beta_q)$$

- 여기서 clip()는:

$$\mathrm{clip}(x, l, u) = \begin{cases} l & \text{if } x < l \\ x & \text{if } l \leq x \leq u \\ u & \text{if } x > u \end{cases}$$

Quantization Exercise에서 연습해보세요!

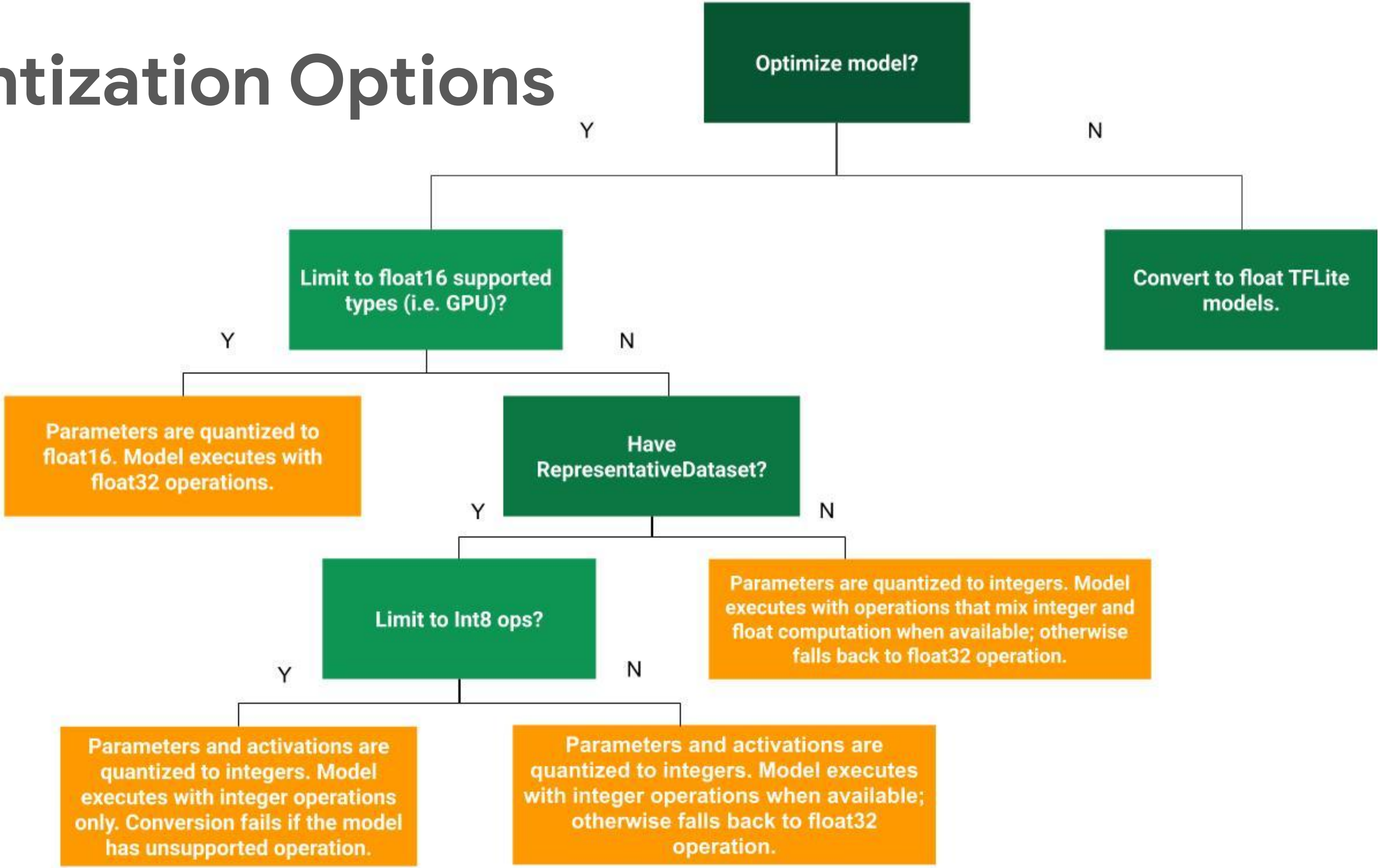
Post Training Quantization Example

```
converter = tf.lite.TFLiteConverter.from_keras_model(base_model)

converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.target_spec.supported_types = [tf.float16]

quantized_model = converter.convert()
```

Quantization Options



Quantization Results

Dynamic Range Quant Colab

Approach	Technique	Size Reduction	Accuracy Loss
Post Training	Float16 Quantization	Up to 50%	Insignificant
	Dynamic Range Quantization (8bit)	Up to 75%	Low-Medium
	Integer Quantization (8bit)	Up to 75%	Medium
During Training (Quantization Aware Training)		Up to 75%	Low

Smaller models ↓
Lower accuracy

Dynamic Range Quantization Example

To quantize the model on export, set the `optimizations` flag to optimize for size:

```
[7] converter.optimizations = [tf.lite.Optimize.DEFAULT]
    tflite_quant_model = converter.convert()
    tflite_model_quant_file = tflite_models_dir/"mnist_model_quant.tflite"
    tflite_model_quant_file.write_bytes(tflite_quant_model)
```

```
INFO:tensorflow:Assets written to: /tmp/tmpka8770dr/assets
INFO:tensorflow:Assets written to: /tmp/tmpka8770dr/assets
WARNING:absl:Buffer deduplication procedure will be skipped when flatbuffer library is
23920
```

Note how the resulting file, is approximately `1/4` the size.

```
!ls -lh {tflite_models_dir}
```

```
total 108K
-rw-r--r-- 1 root root 24K Apr 15 14:35 mnist_model_quant.tflite
-rw-r--r-- 1 root root 83K Apr 15 14:35 mnist_model.tflite
```

+ Code + Text

Dynamic Range Quantization Example

```
    for index in range(len(prediction_digits)):
      if prediction_digits[index] == test_labels[index]:
        accurate_count += 1
    accuracy = accurate_count * 1.0 / len(prediction_digits)

    return accuracy
```

```
print(evaluate_model(interpreter))
```

```
0.9603
```

+ Code

Repeat the evaluation on the dynamic range quantized model to obtain:

```
[15] print(evaluate_model(interpreter_quant))
```

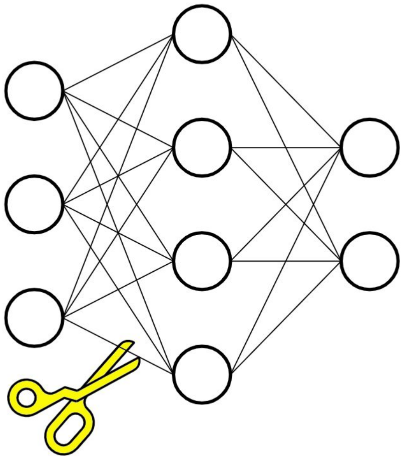
```
0.9598
```

In this example, the compressed model has no difference in the accuracy.

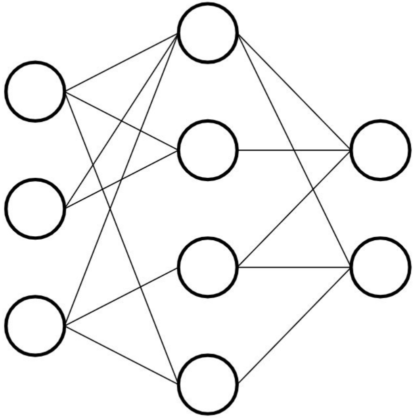
Weight Pruning (aka Sparsity)

As a training time technique, removes unnecessary connections between the layers of a neural network

Results in sparse weight tensors



Before pruning



After pruning

3	2	7	4
9	6	3	8
4	4	1	3
2	3	2	5

Before pruning

0	2	0	4
0	6	3	0
4	0	0	3
0	3	0	5

After pruning

Pruning Results

"Every 4 elements, 2 zero entries"
Nvidia A100

Image Classification

Model	Non-sparse Top-1 Accuracy	Random Sparse Accuracy	Random Sparsity	Structured Sparse Accuracy	Structured Sparsity
InceptionV3	78.1%	78.0%	50%	75.8%	2 by 4
		76.1%	75%		
		74.6%	87.5%		
MobilenetV1 224	71.04%	70.84%	50%	67.35%	2 by 4
MobilenetV2 224	71.77%	69.64%	50%	66.75%	2 by 4

Smaller models ➔ Lower accuracy

Weight Pruning (aka Sparsity)

```
# Train a model with pruning
pruned_model = tfmot.sparsity.keras.prune_low_magnitude(base_model)
callbacks = [ tfmot.sparsity.keras.UpdatePruningStep() ]
pruned_model.compile(...)
pruned_model.fit(x_train, y_train, callbacks=callbacks)
final_model = tfmot.sparsity.keras.strip_pruning(pruned_model)

# Convert to TFLite
converter = tf.lite.TFLiteConverter.from_keras_model(final_model)
converter.optimizations = [tf.lite.Optimize.EXPERIMENTAL_SPARSITY]
tflite_model = converter.convert()
```

Benefits of Weight Pruning

Validated across prediction tasks (image, speech, etc.)

Speed-ups in CPU, and some ML accelerators

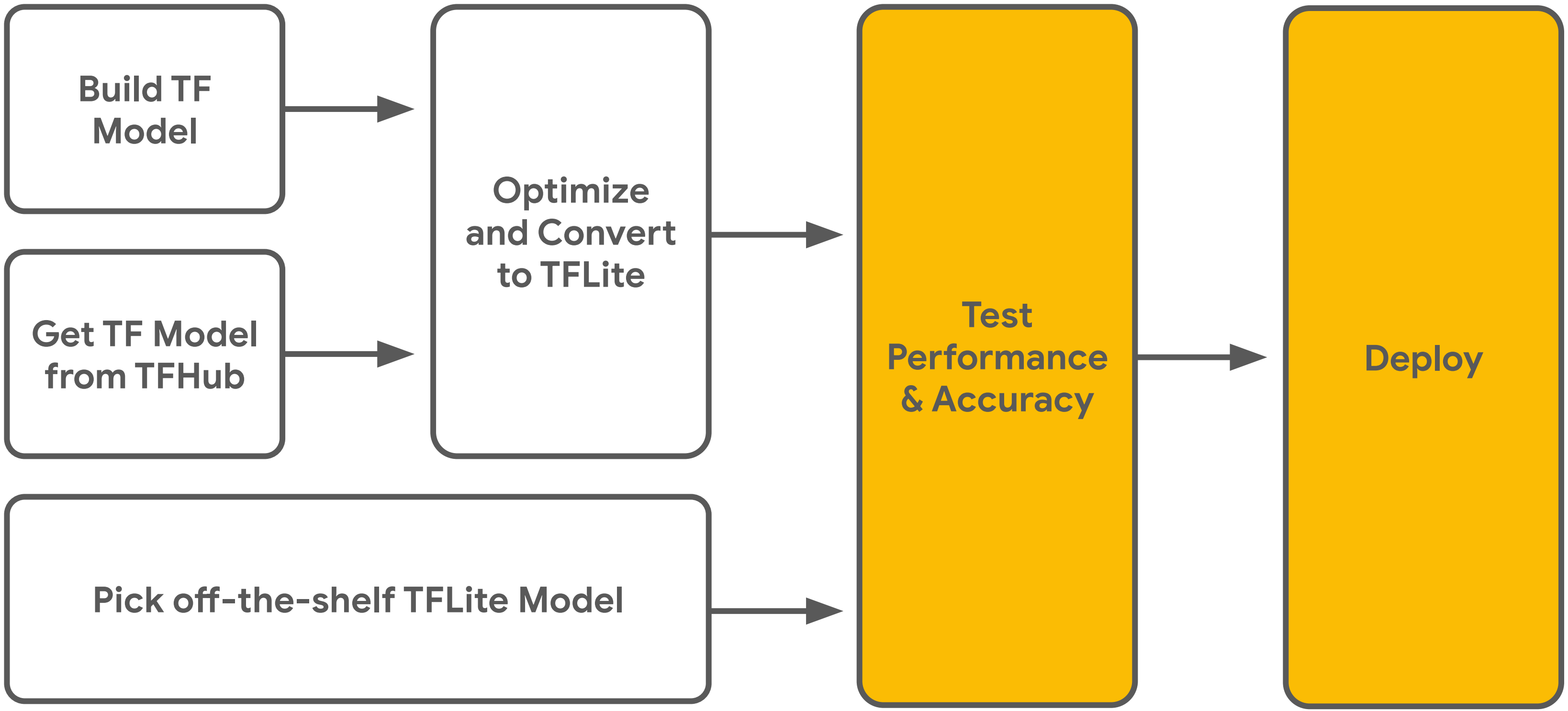
50-90% sparsity, negligible accuracy loss

Works with quantization

Colab on Pruning

Colab on Pruning + Quantization

Workflow at a glance



To get the Best Performance

Choose the best model

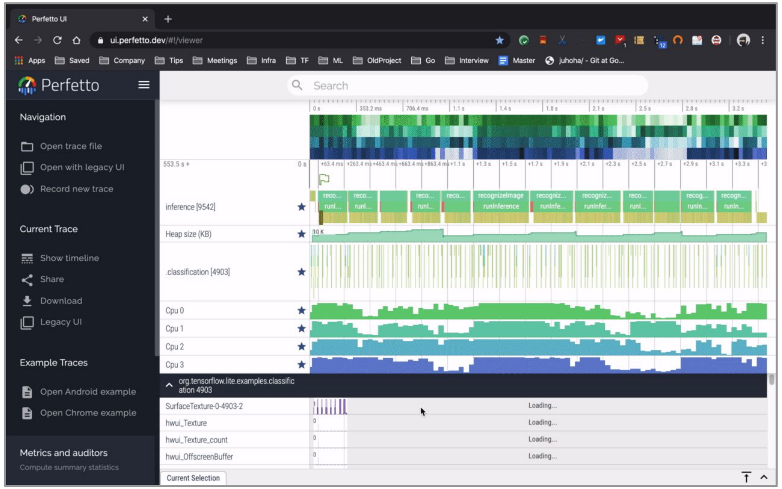
- Choose heavy vs. light models based on accuracy constraint
- Profile and benchmark graph execution
- Apply optimization techniques like quantization, sparsity, etc

Choose the best runtime configuration

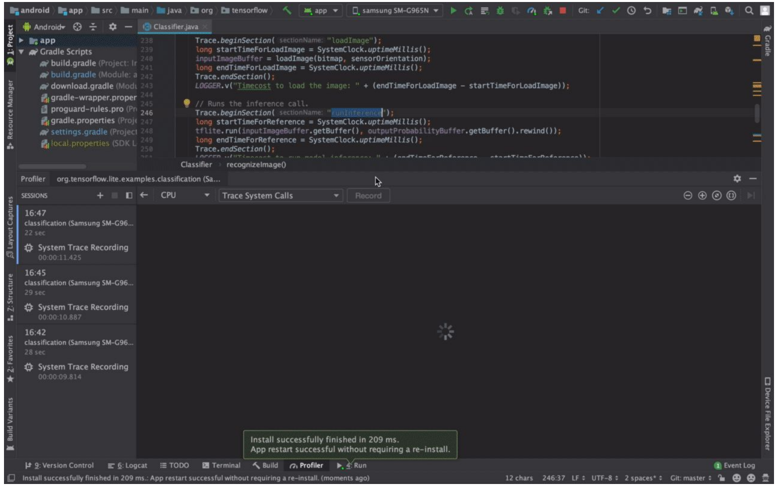
- Try multi-threaded inference
- Evaluate hardware acceleration

Android Profiler

Perfetto
De-facto profiler for Android 10+



Android Studio



Inspect overall + op-level TFLite model profiling (CPU, GPU delegation..)

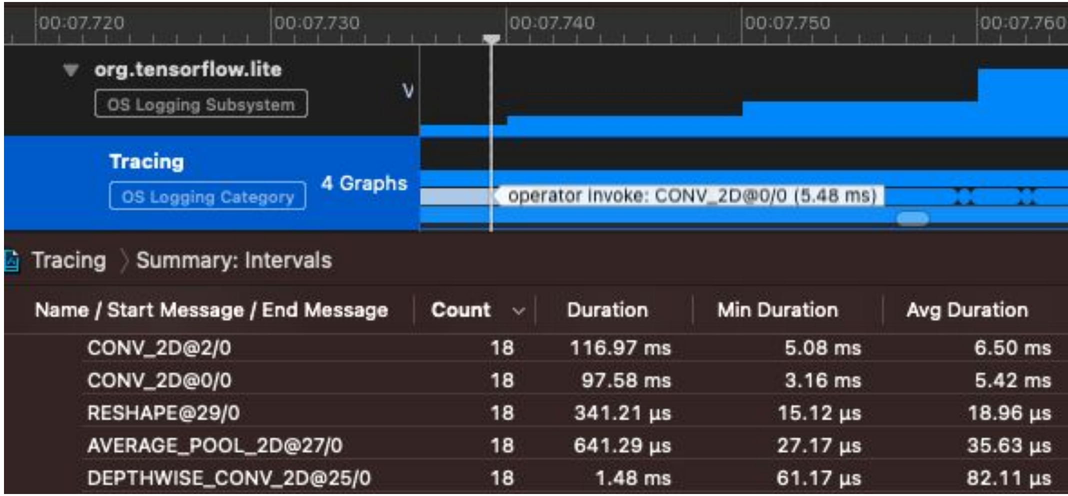
iOS Profiler

Xcode Profiler Integration with Signposts

Build with `debug.tflite.trace` option enabled

Run Xcode Profiler and record os_signpost events

Stop recording and analyze detailed per-op profiling results



Benchmark Tool

TFLite Command Line Benchmark Tool

```
adb shell /data/local/tmp/benchmark_model \
    --graph=/data/local/tmp/test_model.tflite \
    --enable_op_profiling=true \
    --num_threads=4 \
    --use_gpu=true
```

On-device Machine Learning

g.co/on-device-ml

Learn about

the benefits of on-device ML ▪ which solutions fit your needs ▪ how to quickly get started ▪ real-world use cases

TensorFlow Lite

TensorFlow.js

TensorFlow Hub

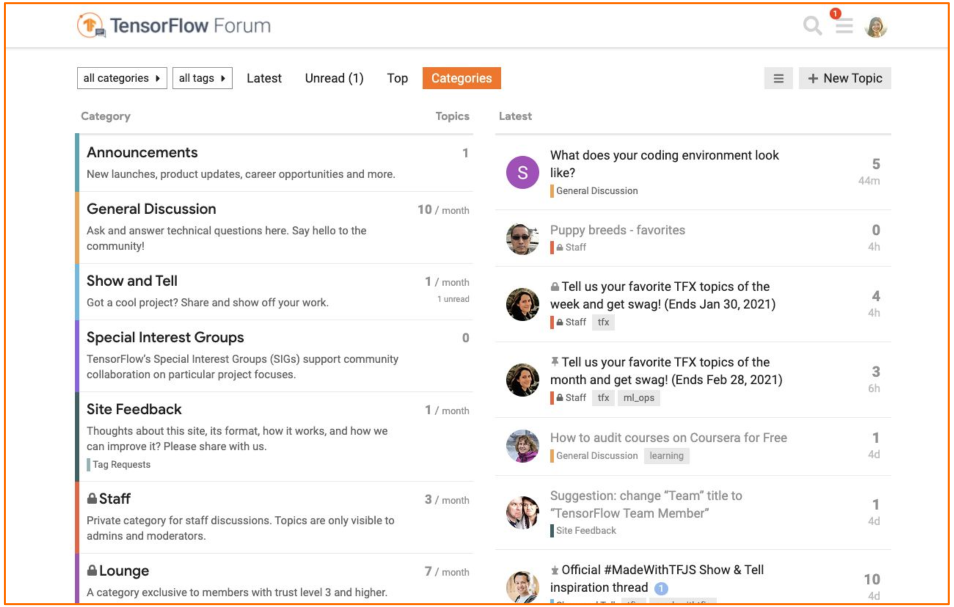
Android ML

ML Kit

MediaPipe

Firebase

Let’s continue the conversation at our newly launched TensorFlow Forum!



discuss.tensorflow.org

A place for constructive conversation, support, inspiration, and sharing of best practices between the TensorFlow community.

Create your account and join the conversation!

Learn more about On-Device ML

To learn more about all of Google's on-device machine learning solutions, visit g.co/on-device-ml

- What are the benefits of ODML?
- Which solution fits my needs?
- Learning paths to get started
- Real-world use cases

TensorFlow Lite

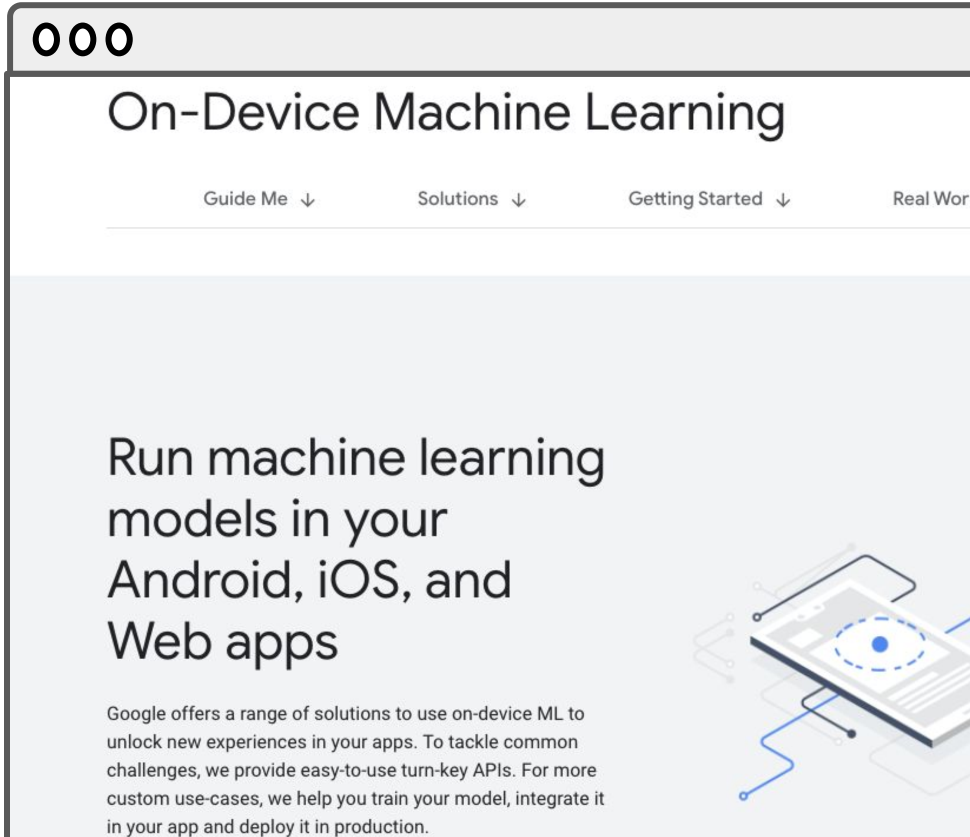
ML Kit

TensorFlow.js

MediaPipe

TensorFlow Hub

Firebase



Thank you!

Resources

tensorflow.org/lite

g.co/on-device-ml

github.com/tensorflow/tensorflow/tree/master/tensorflow/lite

tensorflow.org/model_optimization

github.com/tensorflow/model-optimization